SCALABLE CPU ERROR RECORDER

Background of the Invention

[0001] This invention relates to computer error handling. More specifically, the invention relates to a firmware-based error handling mechanism to support the creation, storage and retrieval of customized and extendible error records in computer platforms.

[0002] Modern computers are designed to monitor their own performance and frequently to test themselves to assure that operations have been performed properly. When a fault occurs, a machine interrupt typically is issued, and the hardware and software attempt to locate and identify the error. Depending on the severity of the error, control programs may shut down the entire machine, may avoid use of the faulty component, or may simply record the fact that an error has occurred.

[0003] System error detection, containment and recovery are critical elements of highly reliable and fault tolerant computing environments. While error detection is primarily accomplished through hardware mechanisms, system software plays a greater role in the containment and recovery of errors. The degree to which overall error handling is effective in maintaining system integrity depends upon the level of coordination and cooperation between the system CPUs, platform hardware fabric, and system software. Vendors of such computer systems therefore have developed maintenance and diagnostic facilities as part of their computer platforms. When a system failure occurs, diagnostic software may attempt to determine the cause of the

failure and may also attempt to store information describing the failure, so that subsequent efforts to resolve or eliminate the failure may benefit from the stored information.

[0004] In the prior art, software-based error handling mechanisms of the type described have traditionally resided in a portion of the computer operating system. As a result, operating system designers have been required to develop unique error handling subsystems for each supported computer platform. Because of this constraint, computer error handling capabilities have been relatively limited in the prior art. In particular, designers of multiple computing environments have been forced to isolate the error management functions of each component operating system. Similarly, designers of complex computer platforms having multiple domains and/or partitions have been forced to deploy separate and isolated error management systems. Additionally, Original Equipment Manufacturers (OEMs) have been restricted in their ability to develop customized computer platforms that provide enhanced maintenance capabilities.

[0005] Accordingly, there is a need in the art for a unified and standardized approach to computer error handling at the firmware level, outside the traditional sphere of an operating system. Such an error handling mechanism would allow computer platform designers and operating system engineers to develop standard error management subsystems that make effective use of common interfaces and methods. A standard error handling mechanism would also permit OEMs to develop error parsers, utilities and enhanced maintenance diagnostics that do not depend on the specific features any particular operating system.

Brief Description Of The Drawings

[0006] FIG. 1 is an abstract functional block diagram of a computer system incorporating a scalable error logging mechanism in accordance with an embodiment of the present invention.

[0007] FIG. 2 is a diagram of the variable-length error record data structure employed by an embodiment of the present invention.

[0008] FIG. 3 is a flow diagram illustrating the error handling control logic according to an embodiment of the present invention.

[0009] FIG. 4 is a flow diagram illustrating the error record retrieval control logic in accordance with an embodiment of the present invention.

[0010] FIG. 5 is a table illustrating a method employed by an embodiment of the present invention to maintain the consumed/unconsumed status of error records.

Detailed Description

[0011] Embodiments of the present invention provide a firmware mechanism for creating, storing and retrieving variable-length records associated with error events occurring in a computer platform. According to an embodiment, the mechanism responds to error notifications by invoking a firmware-based error-handling module. This error-handling module may retrieve processor-specific error information and also may interrogate other components of the computer platform to determine their error status. Then, according to the nature of the discovered errors, the error-handling module may assemble the retrieved error information and status information into a variable-length error record which the error-handling module may then store in a memory. Upon request from an external processing agent, the error-handling module may retrieve a previously-stored error record and present it to the requesting agent.

[0012] Referring now in detail to the drawings, wherein like parts are designated by like reference numerals throughout, there is illustrated in FIG. 1 an abstract functional block diagram of a computer system 100 incorporating a scalable error logging mechanism in accordance with an embodiment of the present invention. The computer system 100 may exist in isolation, or it may be one component of a partitioned, multi-processor system. The functional elements of the computer system 100 may include an operating system 110 that accesses computer resources via system firmware 120 and processor firmware 130. As is known in the art, firmware corresponds to computer instructions that are stored in non-volatile memory, i.e., memory that holds its contents in the absence of external power. Traditionally, computer designers provide firmware to perform various low-level tasks that are basic to the operation of a computer system or which exhibit a critical nature. Firmware also may be provided in several layers of abstraction that are equivalent to different levels of responsibility or priority. In addition

to being readily available for execution by a processor, firmware may also be protected from inadvertent corruption if it is stored in read-only memory.

[0013] In FIG. 1, two levels of firmware are shown. Processor firmware 130 may provide functionality associated with processor management (e.g., saving and restoring processor states), while system firmware 120 may provide broader capabilities relating to platform management and resource management (e.g., device error handling and multi-processor coordination). System firmware 120 may also supply platform-specific versions of common operating system functions. Thus, because system firmware 120 may reside at a higher level of abstraction, it may access processor firmware 130, system memory 140, and computer platform devices 160 (which include peripheral devices). To manage errors occurring at the platform level, system firmware 120 may incorporate an error handler 125 that manages error processing at a level of abstraction above the processor 150, but below the level of operating system 110. Processor 150 is the traditional processing unit of the computer system. Processor 150 may communicate with processor firmware 130 as well as with system memory 140 and computer platform devices 160, in order to execute the primary operations of the computer system. System memory 140 includes the working random-access memory (RAM) of the computer system as well as non-volatile memory 145, which together hold instructions and data used by processor 150 and computer platform devices 160.

[0014] Continuing to refer to FIG. 1, a computer system error may arise from one of the computer platform devices 160, from the processor 150, or from the processor firmware 130. When a computer system error occurs, error handler 125 may be invoked. Error handler 125 may comprise a collection of computer instructions that when executed, cause error events to be discovered and recorded for later analysis. Error handler 125 may also comprise computer instructions that attempt to resolve and/or correct an outstanding error to allow a computer system to continue processing. Invocation of error handler 125 may take place indirectly as a result of an interrupt (either a hardware interrupt or a software interrupt), or the error handler may be invoked directly by the operating system 110 through a polling mechanism. Upon invocation, the error handler 125 may extract device-level error information by reading the various hardware registers of the computer platform devices 160 which are within the scope of the processor 150. Alternatively, the processor firmware 130 may extract

the device-level error information from the computer platform devices 160 and may then present the collected error information to the error handler 125. Once all the device-level error information has been collected, the error handler 125 may format the collected error information into a variable-length error record which is then stored in system memory. According to an embodiment, error records are stored in non-volatile memory 145 to preserve the error information in the event that the computer is restarted or shut down before the error information could be stored elsewhere.

[0015] FIG. 2 is a diagram of the variable-length error record data structure employed by the error handler 125 (FIG. 1) according an embodiment of the present invention. Each error record 200 captures error information for a single error event. However, because there may be multiple device errors associated with a single error event, the error record 200 includes a variable number of sections, each of which may store error information pertaining to a particular error device. An error record 200 may consist of a generic record header 210 followed by a list of sections containing actual error information for the error event, each section containing information relating to a particular device (for example, a processor, platform memory, a platform bus, etc.) that may have contributed to the error condition. Each section of error record 200 may have two parts: a section header and a section body. Thus, with reference to FIG. 2, the first section may have a section-1 header 220 and a section-1 body 230; the second section (if present), may have a section-2 header 240 and a section-2 body 250; and the last section (if present) may have a section-n header 260 and a section-n body 270. The overall size of an error record 200 is indicated by a record length field, which is an element of record header 210. The record length field may be dynamically set by the error handler 125 (FIG. 1) based on the total size of all the section headers and section bodies combined.

[0016] According to an embodiment, the format of the record header 210 may include the following fields:

Field	Length	Description
RECORD_ID	8 bytes	Unique monotonically increasing identifier.
REVISION	2 bytes	Major and minor revision number of the error record in BCD format. Byte 0 – Minor number Byte 1 – Major number
ERR_SEVERITY	1 byte	0 – Recoverable

		1 – Fatal 2 – Corrected
VALIDATION_BITS	1 byte	If Bit 0 = 1, the OEM_PLATFORM_ID field contains valid information.
RECORD_LEN	4 bytes	Length of the record in bytes, including the header.
TIME_STAMP	8 bytes	Time of the error. Byte 0 – seconds Byte 1 – minutes Byte 2 – hours Byte 3 – reserved Byte 4 – day Byte 5 – month Byte 6 – year Byte 7 – century
OEM_PLATFORM_ID	16 bytes	Unique identifier of the OEM platform.

[0017] Still referring to FIG. 2, a series of device-specific sections may follow record header 210. Each section may have a section header and a section body. The format of a section body may depend on its corresponding device and thus is peculiar to specific implementations. However, the section header may be standardized. According to an embodiment, the format of the section header may include the following fields:

Field	Length	Description
GUID	16 bytes	Global Unique Identifier corresponding to the specific device.
REVISION	2 bytes	Major and minor revision number of the section in BCD format. Byte 0 – Minor number Byte 1 – Major number
ERROR_RECOVERY_INFO	1 byte	If Bit 7 = 1, remaining bits contain information about the error. Bit 6-3 = Reserved (must be zero). Bit 2 = Reset. If set, the component must be re-initialized or re-enabled by the operating system prior to use. Bit 1 = Containment Warning. If set, the error was not contained within the processor or memory hierarchy and the error may have propagated to persistent storage or to a network. If Bit 0 is set, the error has been corrected.
RESERVED	1 byte	Reserved.
SECTION_LEN	4 bytes	Length of the error device section in bytes, including the header.

[0018] An error record 200 may be stored in system memory 140 (FIG. 1) or in non-volatile memory 145 (FIG. 1). The error record 200 storage and indexing scheme may employ simple tables containing pointers to the error records 200, linked lists of pointers to the error records 200, or other methods known in the art for storing, searching and retrieving variable-length data structures.

[0019] FIG. 3 is a flow diagram illustrating operation of the error handling control logic according to an embodiment of the present invention. The operation may begin when one of the computer platform devices 160 (FIG. 1) generates a hardware or software interrupt corresponding to an error condition. Alternatively, the operation may begin when the operating system 110 (FIG. 1) invokes the error handler 125 (FIG. 1) through a polling mechanism. If there is an outstanding error condition (310), the error handler may then collect error information (320) by querying the various registers of the computer platform devices 160 (FIG. 1). According to alternate embodiment, the processor firmware 130 (FIG. 1) may extract some or all of the device-level error information from the computer platform devices 160 and may then present the collected error information to the error handler 125. Once all of the error information has been collected (320), the error handler may format the information into an error record (330) and then store the error record in system memory (340). If stored error records are intended to be accessed following a computer system re-boot or restart, the error handler may store error records in non-volatile memory. Finally, after the error record has been stored in memory, the error handler may mark the error record as unconsumed (350) until the error record is retrieved by the operating system or by some other error management capability.

[0020] FIG. 4 is a flow diagram illustrating the error record retrieval control logic in accordance with an embodiment of the present invention. The operation may begin when the operating system 110 (FIG. 1) or other error management capability invokes the error handler 125 (FIG. 1) to retrieve an error record. An error record may be retrieved by its identifier or it may be retrieved according to an ordering scheme such as first-in-first-out (FIFO) or last-in-first-out (LIFO). When requested, the error handler may retrieve the appropriate error record (410) and may provide the error record to the requesting agent. The error handler may then mark the error record as consumed (420), so that the memory associated with the consumed error record may be reused at a later time. After an error record has been marked as consumed, the error handler

may then perform a garbage collection algorithm to collect and combine the memory associated with consumed error records. Garbage collection algorithms are well-known in the art.

[0021] FIG. 5 is a table illustrating a method employed by an embodiment of the present invention to maintain the consumed/unconsumed status of error records. The error handler 125 (FIG. 1) may construct a consumed record table 500, which contains information corresponding to each error record. For each error record in the table, the error handler may maintain an ID Number 510 and a Consumed indicator 520. When the error handler stores a new error record (see FIG. 3, item 340), the error handler may simultaneously create a new entry in the consumed record table 500 and may indicate that the record has not been consumed. At a later time, when the error handler is requested to retrieve an error record (see FIG. 4, item 410), the error handler may locate the error record in the consumed record table 500 and may mark the error record as consumed.

[0022] According to one embodiment, the present invention provides a firmware-based mechanism for creating, storing and retrieving variable-length records associated with error events occurring in a single-processor computing system. According to another embodiment, the present invention may also operate in a multi-processor environment in which processors are partitioned into domains, each having dedicated computer platform devices. In such a multi-processor environment, an instance of the present invention may exist and operate within the domain of each individual processor. Upon receiving an error notification, an instance of this embodiment may interrogate the components of the computer platform lying within the scope of the embodiment's processor domain to discover the outstanding error events and construct an appropriate error record.

[0023] Several embodiments of the present invention are specifically illustrated and described herein. However, it will be appreciated that modifications and variations of the present invention are covered by the above teachings and within the purview of the appended claims without departing from the spirit and intended scope of the invention.